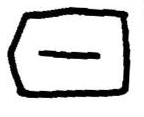
# K-means算法

## 1.K-means算法基本思路

K-means算法[[1]](#footnote-0)又称K-均值算法，属于聚类算法的一种，最早提出与1967年，其出现的目的就是为了解决聚类问题，是应用最广泛的聚类算法之一。

所谓聚类算法，既是首先决定几个“簇”（可以理解为需要把未知的数据划分成几个类型），然后根据计算每一组数据并将相似度较高的数据对象划分为同一簇。最后通过最小化**簇内误差平方和（WCSS）[[2]](#footnote-1)**来进行模型的优化。

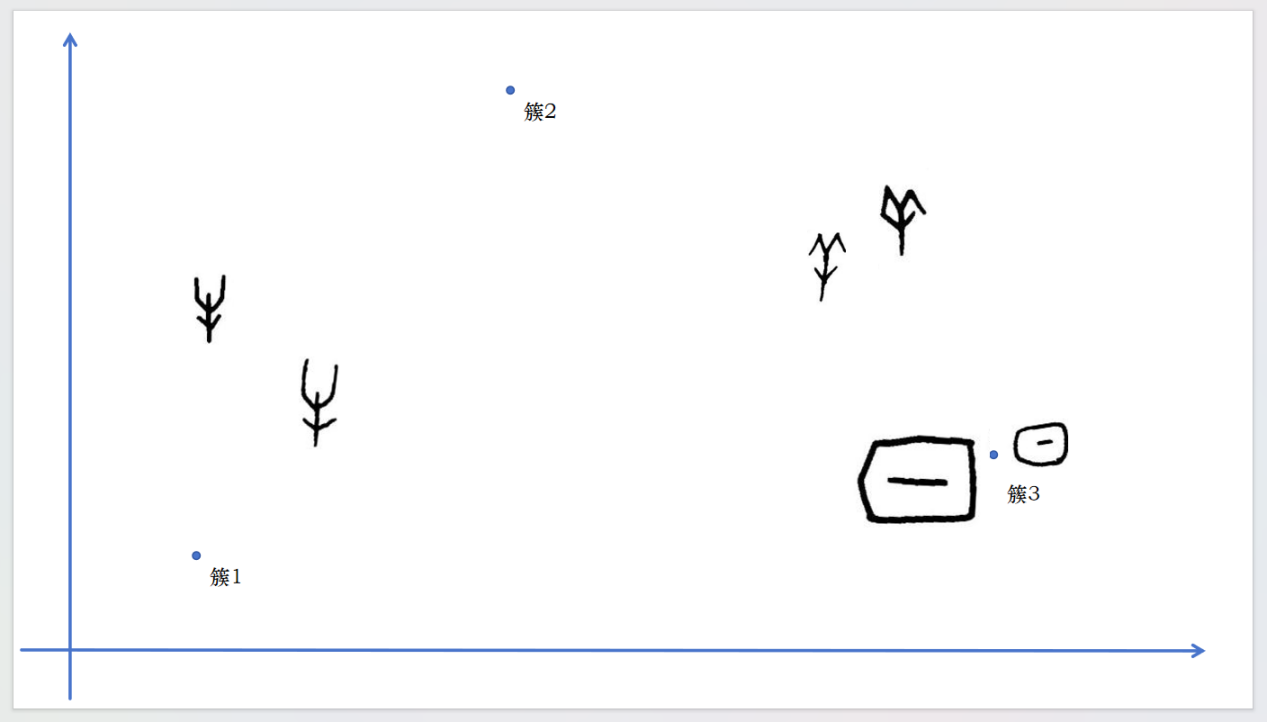
假设我们训练集有如下六张甲骨字图片：

aoaipuy20h_h00331_386_388_59_118 aoaipuy20h_h00884_127_229_52_107 66iowpdbuc_h00039_117_187_44_80 66iowpdbuc_h01615_271_152_43_97  p8w7ujqanz_h03932_107_114_63_51

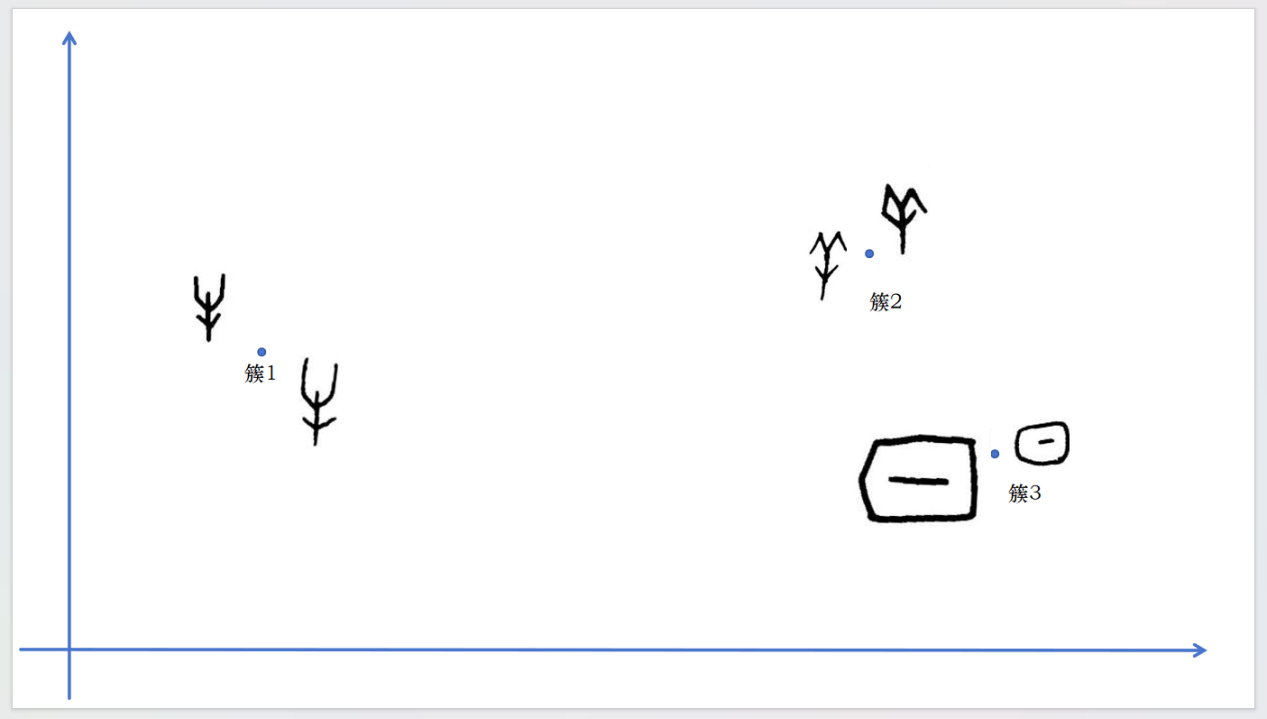
我们预设的簇的个数为3，假设其在坐标系中表示如下[[3]](#footnote-2)：



然后经过计算我们将图片进行相似分析并映射在坐标系中，如下图：



我们可以看到部分簇点已经偏离了质心[[4]](#footnote-3)，所以我们要对簇点进行更新，如下图：



以上示例简单展示了一下K-means算法的基本过程，并不完全但也能做到管中窥豹了，接下来我们将从数学角度来详细讲解这个算法并进行手算推导，以便我们更好理解该算法。

## 2.K-means算法的数学原理

### 2.1 K-means算法中的基本定义

在开始介绍K-means算法数学理论之前，我们需要定义几个将会使用到的量。假设有个数据，这些数据是独立的概率均为p且这些数据均满足，其中代表一个N维的欧几里得空间。

我们定义k个中心点的合集为，其中每个中心点.对于中心点的定义是为了明确K-means算法的核心目标，即通过优化这些中心点的位置来实现数据的有效聚类。

对于给定的中心点集合X，分区定义为：

即每个数据点被分配到距离其最近的中心点所对应的簇中。通过定义分区，明确每个数据点所属的簇，从而实现聚类的基本操作。这一定义是K-means算法中“分配步骤”的基础。

### 2.2 K-means目标函数

K-means算法的核心目标是最小化类内平方误差，其对应的优化目标函数为：

以上为基于**概率分布**的连续形式，适用于理论分析。

以上为基于**离散数据点**的有限样本形式，适用于实际计算，在K-means算法中，我们一般以基于离散数据点有限样本形式计算模型误差。

通过优化这一目标函数，我们可以逐步优化模型直至达到最优，我们接下来将讨论这一目标函数的来源。

#### 2.2.1 经验误差与理论误差

经验误差是基于有限样本数据计算得到的误差，衡量算法在实际训练数据上的表现，是我们在有限数据集上直接观察到的误差。在K-means中，经验误差的定义源于K-means算法的核心优化目标，即最小化簇内数据点到其簇中心的距离平方和。这一目标在数学上可以表达为：

为了将这一目标量化为一个易于分析的形式，我们引入经验误差作为衡量簇内紧密度的指标。具体来说，经验误差定义为所有数据点到其所属簇中心的距离平方和的平均值：

通过平均化所有数据点的最小距离平方，经验误差提供了一个标准化的指标，用于评估当前簇划分的质量。较低的经验误差表示簇内数据点更加紧密，有助于算法在每次迭代中朝着更优的簇划分方向前进。

**理论误差**（即为泛化误差）的定义源于统计学习理论，旨在描述算法在整体数据分布上的理想表现。与经验误差不同，理论误差考虑的是所有可能数据点（即整个数据分布）的期望距离平方和：

理论误差提供了一个衡量算法在整个数据分布上的聚类质量的标准。通过将理论误差与经验误差相比较，我们可以评估算法在样本数据集上的表现是否能够良好地推广到整体数据分布。

#### 2.2.2 随机变量的收敛性证明

根据大数定律，当N趋于无穷时，经验误差作为独立同分布随机变量的样本均值，将几乎必然收敛于其期望值.

因此我们可得式[[5]](#footnote-4)：

#### 2.2.3 误差的渐进行为证明

定义分区误差，其中是分区的理论中心点，是经验中心点。对于每个簇i，随着样本数N趋于无穷，经验中心点几乎必然收敛于理论中心点，即：

根据大数定律，对于有限个簇k，有：

这表明随着迭代次数m的增加，误差的平均值趋于零。整理可得：

#### 2.2.4 目标函数的推导

根据2.2.2节我们可得式：

经验误差表示为所有数据点的最小距离平方和的平均值。因此，整体的误差函数为：

随着样本，根据2.2.2我们可知，经验误差收敛于理论误差，即：

根据2.2.3节我们可知，在大样本条件下，有限样本误差的平均值趋于零，即：

这意味着，随着样本数量的增加，经验误差与理论误差之间的差异趋于零，从而保证了误差函数的稳定性与一致性。

结合2.2.2与2.2.3节得到的结论，我们可以将K-means算法的目标函数写为以下形式：

### 2.3 K-means的优化

K-means算法的优化和更新过程是通过迭代进行的，目的是不断改进簇中心的位置，以减少簇内的误差（即样本点到簇中心的距离平方和）。这个过程包含两个主要步骤：簇的分配（Assignment Step）和簇中心的更新（Update Step）。

#### 2.3.1 簇的分配

在每一次迭代中，K-means算法首先对每个数据点进行分配。每个数据点被分配到离它最近的簇中心所代表的簇。具体地说，对于数据集中的每个样本点，计算该点到所有簇中心的距离，选择距离最小的簇中心作为该样本点的簇归属。形式化表示为：

其中，代表样本点，是样本点所归属的簇。

#### 2.3.2 簇中心的更新

在完成簇的分配之后，下一步是重新计算每个簇的簇中心。新的簇中心是当前簇中所有点的均值（即质心），计算公式如下：

其中，是簇中样本点的数量。

## 3.K-means的手算推理

我们的训练集如下：

Filename,Feature\_1,Feature\_2,Feature\_3,Feature\_4,Feature\_5,Feature\_6,Feature\_7,Feature\_8,Feature\_9,Label

\_勿1.png,0.72,0.63,0.67,0.65,0.59,0.41,0.99,0.69,0.82,勿

\_勿2.png,0.66,0.82,0.98,0.69,0.84,0.86,0.81,0.79,0.91,勿

\_牛1.png,0.78,0.98,0.76,0.69,0.41,0.88,0.93,0.58,1.0,牛

\_牛2.png,0.74,0.77,0.75,0.85,0.41,0.92,0.88,0.63,0.93,牛

\_羊1.png,0.82,0.66,0.73,0.91,0.55,0.95,1.0,0.78,1.0,羊

\_羊2.png,0.74,0.69,0.74,0.88,0.5,0.89,0.82,0.8,0.92,羊

测试集如下：

Feature\_1,Feature\_2,Feature\_3,Feature\_4,Feature\_5,Feature\_6,Feature\_7,Feature\_8,Feature\_9

0.72,0.58,0.63,0.93,0.36,0.91,1.0,0.69,1.0

接下来我们将利用训练集与预测集来进行K-means算法的推理，为了阅读的方便，我们在此会保留标签。在此之前我们再来回顾一下K-means算法流程：

1. 初始化：选择K个初始中心点；
2. 分配：将每个数据点分配到距离其最近的中心点所属的簇。
3. 更新：重新计算每个簇的中心点，作为该簇所有点的均值。
4. 迭代：重复分配和更新步骤，直到中心点不再变化或达到预设的迭代次数。

### 3.1 初始化

初始化簇点[[6]](#footnote-5)：

Cluster 1 (C1): \_勿1.png

Cluster 2 (C2): \_勿2.png

Cluster 3 (C3): \_羊1.png

即：

C1: [0.72,0.63,0.67,0.65,0.59,0.41,0.99,0.69,0.82]

C2: [0.66,0.82,0.98,0.69,0.84,0.86,0.81,0.79,0.91]

C3: [0.82,0.66,0.73,0.91,0.55,0.95,1.00,0.78,1.00]

### 3.2 第一次分配

计算每个训练数据点到各簇中心的欧几里得距离的平方，并进行分配，公式为：

1.\_勿1.png:

距离C1：

距离C2：

距离C3：

综上，\_勿1.png被分配到C1.

2.\_勿2.png：

距离C1：

距离C2：

距离C3：

综上，\_勿2.png被分配到C2.

3.\_牛1.png：

距离C1;

距离C2;

距离C3：

综上，\_牛1.png被分配到C2；

4.\_牛2.png：

距离C1：

距离C2：

距离C3：

综上，.\_牛2.png被分配到C2.

5.\_羊1.png：

距离C1：

距离C2：

距离C3：

综上，\_羊1.png分配到C3。

6.\_羊2.png：

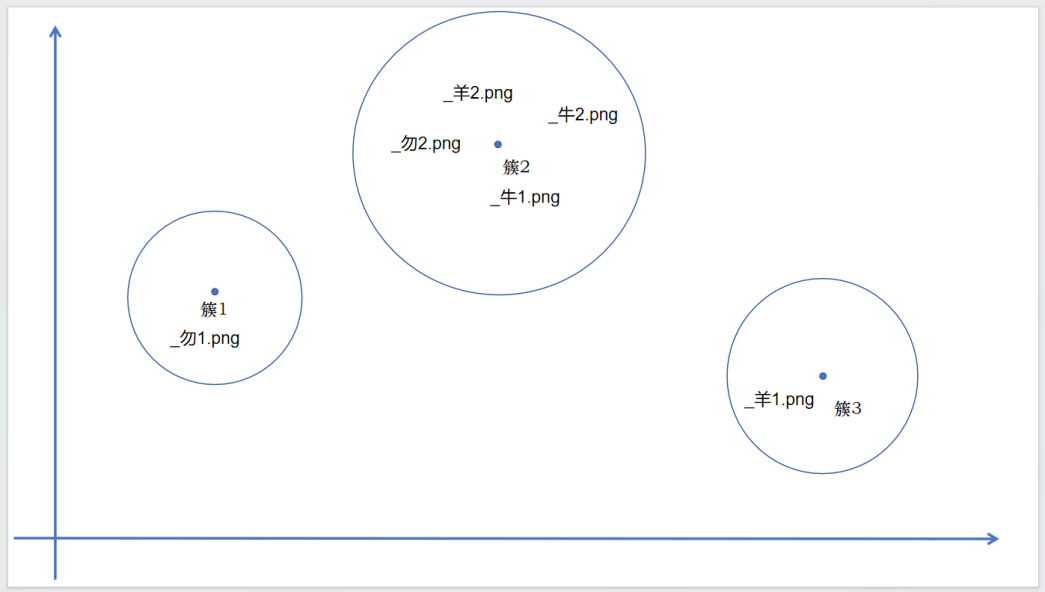
距离C1：

距离C2：

距离C3：

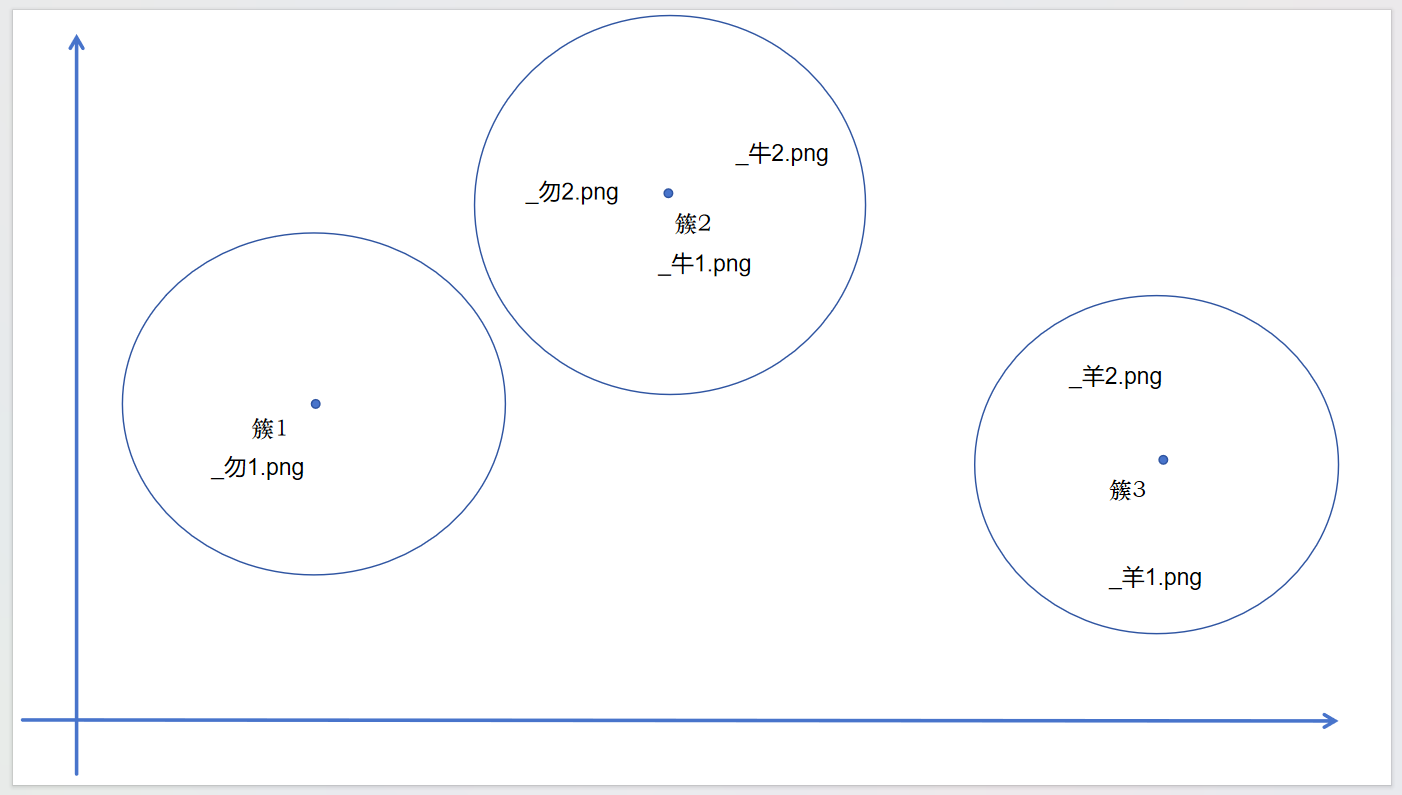
综上，\_羊2.png被分配到C2.

根据以上的计算结果，我们可以得到如下图中的关系（仅表示数据与簇的关系）：



### 3.3 迭代更新与误差计算

根据新分配的簇与数据，我们重新计算三个簇的质心，然后根据新的簇的质心再一次将数据进行更新，知道每一个数据所属于的簇不再变化时，我们认为此时模型已收敛，最终得到的模型如下图：



最终模型得到的簇的质心为：

C1：[0.72, 0.63, 0.67, 0.65, 0.59, 0.41, 0.99, 0.69, 0.82]

C2：[0.73, 0.81, 0.83, 0.7433, 0.5533, 0.8933, 0.8733, 0.6667, 0.947]

C3：[0.78, 0.675, 0.735, 0.895, 0.525, 0.92, 0.91, 0.79, 0.96]

结合最新的簇的质心，我们可以对预模型进行误差计算，公式：

1.Cluster 1 (C1): \_勿1.png

2.Cluster 2 (C2): \_勿2.png, \_牛1.png, \_牛2.png

\_勿2.png:

\_牛1.png:

\_牛2.png

综上，簇2的误差为：

3.Cluster 3 (C3): \_羊1.png, \_羊2.png

\_羊1.png:

\_羊2.png:

综上所述，簇3的总误差为：

根据以上三个簇的误差进行求和，我们可以得到模型的总误差为：

### 3.4 使用模型进行预测

已知我们的测试数据点为：

带入模型进行计算：

1.距离到:

2.距离到：

3.距离到：

易得：

综上，我们可知预测数据的分类应该属于簇3。

## 4.K-means代码示例

import os

import pandas as pd

from tqdm import tqdm

import torch

from torch.utils.data import Dataset, DataLoader

from torchvision import transforms

from PIL import Image

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from collections import defaultdict, Counter

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# 定义图像预处理：调整大小为64x64，二值化，归一化

class Binarize(object):

def \_\_call\_\_(self, tensor):

return (tensor > 0.5).float()

transform = transforms.Compose([

transforms.Resize((64, 64)),

transforms.ToTensor(),

Binarize(),

transforms.Normalize(mean=[0.5], std=[0.5]) # 简单归一化

])

# 自定义数据集

class ImageDataset(Dataset):

def \_\_init\_\_(self, image\_dir, transform=None):

self.image\_paths = []

self.labels = []

self.transform = transform

supported\_extensions = ('.png', '.jpg', '.jpeg', '.bmp', '.gif')

for filename in os.listdir(image\_dir):

if filename.lower().endswith(supported\_extensions):

self.image\_paths.append(os.path.join(image\_dir, filename))

label = filename.split('\_')[0]

self.labels.append(label)

def \_\_len\_\_(self):

return len(self.image\_paths)

def \_\_getitem\_\_(self, idx):

image\_path = self.image\_paths[idx]

try:

image = Image.open(image\_path).convert('L') # 转为灰度图

except Exception as e:

print(f"无法打开图片: {image\_path}, 错误: {e}")

# 返回一个全黑的图片

image = Image.new('L', (64, 64), 0)

if self.transform:

image = self.transform(image)

label = self.labels[idx]

return image, label

def main():

# 设置设备为GPU（设备号0），如果不可用则使用CPU

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

print(f"使用的设备: {device}")

# 定义路径

all\_images\_folder = 'all\_images' # 确保路径正确

csv\_output = 'cluster\_labels.csv'

report\_output = 'classification\_report.txt'

k = 60 # 簇的数量

# 加载数据

dataset = ImageDataset(all\_images\_folder, transform=transform)

dataloader = DataLoader(dataset, batch\_size=128, shuffle=False, num\_workers=0) # 设置num\_workers=0以避免多进程问题

# 提取特征：将图像展平为一维向量

features = []

labels = []

with torch.no\_grad():

for i, (images, lbls) in enumerate(tqdm(dataloader, desc="提取特征")):

images = images.to(device)

# 将图像展平

batch\_features = images.view(images.size(0), -1)

features.append(batch\_features.cpu().numpy())

labels.extend(lbls)

# 调试信息：打印前两批的特征形状和标签

if i < 2:

print(f"批次 {i + 1} 特征形状: {batch\_features.shape}")

print(f"批次 {i + 1} 标签: {lbls}")

features = np.vstack(features)

print(f"特征提取完成，特征形状: {features.shape}") # 应为 (num\_images, 4096)

# 打印部分特征和标签以确认

if features.shape[0] > 0:

print("示例特征 (第一个样本前10个值):", features[0][:10])

print("示例标签 (第一个样本):", labels[0])

# 将标签转换为数字编码

label\_set = sorted(list(set(labels)))

label\_to\_index = {label: idx for idx, label in enumerate(label\_set)}

index\_to\_label = {idx: label for label, idx in label\_to\_index.items()}

y = np.array([label\_to\_index[label] for label in labels])

# 检查每个类别的样本数

label\_counts = Counter(y)

print("每个类别的样本数:")

for label\_idx, count in label\_counts.items():

print(f"类别 '{index\_to\_label[label\_idx]}' (编码 {label\_idx}): {count} 个样本")

# 识别样本数少于2的类别

singleton\_labels = [label\_idx for label\_idx, count in label\_counts.items() if count < 2]

if singleton\_labels:

print(f"移除以下样本数少于2的类别: {[index\_to\_label[label] for label in singleton\_labels]}")

# 创建掩码，保留样本数>=2的样本

mask = np.isin(y, singleton\_labels, invert=True)

X\_filtered = features[mask]

y\_filtered = y[mask]

print(f"移除后的样本数: {X\_filtered.shape[0]}")

else:

print("所有类别的样本数均满足要求。")

X\_filtered = features

y\_filtered = y

# 检查是否有足够的样本进行划分

if len(y\_filtered) == 0:

print("没有足够的样本进行训练和测试。请检查数据集。")

return

# 划分训练集和测试集

try:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_filtered, y\_filtered, test\_size=0.3, random\_state=42, stratify=y\_filtered)

except ValueError as e:

print(f"数据集划分失败: {e}")

print("尝试不使用分层抽样。")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_filtered, y\_filtered, test\_size=0.3, random\_state=42, stratify=None)

print(f"训练集大小: {X\_train.shape[0]}, 测试集大小: {X\_test.shape[0]}")

# K-Means 聚类

print("在训练集上进行K-Means聚类...")

kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)

kmeans.fit(X\_train)

print("K-Means聚类完成.")

# 为每个簇分配一个类别标签（簇中最多的原始标签）

cluster\_to\_labels = defaultdict(list)

for cluster\_id, label in zip(kmeans.labels\_, y\_train):

cluster\_to\_labels[cluster\_id].append(label)

cluster\_names = {}

for cluster\_id, label\_list in cluster\_to\_labels.items():

if label\_list:

most\_common\_label, count = Counter(label\_list).most\_common(1)[0]

cluster\_names[cluster\_id] = index\_to\_label[most\_common\_label]

else:

cluster\_names[cluster\_id] = "Unknown"

# 将测试集的特征进行聚类预测

test\_clusters = kmeans.predict(X\_test)

# 将簇ID映射为类别标签

y\_pred = [cluster\_names[cluster\_id] for cluster\_id in test\_clusters]

y\_true = [index\_to\_label[label] for label in y\_test]

# 计算精确度

accuracy = accuracy\_score(y\_true, y\_pred)

print(f"分类精确度: {accuracy:.4f}")

# 计算误差率

error\_rate = 1 - accuracy

print(f"分类误差率: {error\_rate:.4f}")

# 计算混淆矩阵（可选）

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

print("混淆矩阵:")

print(conf\_matrix)

# 可视化混淆矩阵

plt.figure(figsize=(12, 10))

sns.heatmap(conf\_matrix, annot=True, fmt='d', xticklabels=label\_set, yticklabels=label\_set, cmap='Blues')

plt.xlabel('预测类别')

plt.ylabel('实际类别')

plt.title('混淆矩阵')

plt.show()

# 创建CSV文件，记录每个簇中不同标签的名称及其包含的图片数目

csv\_data = []

for cluster\_id in range(k):

label\_indices = cluster\_to\_labels.get(cluster\_id, [])

label\_counts\_cluster = Counter(label\_indices)

for label\_idx, count in label\_counts\_cluster.items():

csv\_data.append({

'Cluster\_ID': cluster\_id,

'Cluster\_Name': cluster\_names.get(cluster\_id, "Unknown"),

'Label': index\_to\_label[label\_idx],

'Count': count

})

df = pd.DataFrame(csv\_data)

df.to\_csv(csv\_output, index=False)

print(f"CSV文件已生成: {csv\_output}")

# 生成报告

report = f"""

分类任务使用K-Means算法完成。

总图片数: {len(labels)}

移除样本数少于2的类别后，保留样本数: {len(y\_filtered)}

训练集大小: {X\_train.shape[0]}

测试集大小: {X\_test.shape[0]}

分类精确度: {accuracy:.4f}

分类误差率: {error\_rate:.4f}

生成的CSV文件: {csv\_output}

"""

with open(report\_output, 'w', encoding='utf-8') as f:

f.write(report)

print(f"分类报告已生成: {report\_output}")

if \_\_name\_\_ == '\_\_main\_\_':

main()

以上代码实现了一个基于K-Means聚类算法的图像分类任务。首先，代码加载并预处理图像数据，将其转换为灰度图并展平为特征向量。然后，筛选出样本数大于等于2的类别，并使用train\_test\_split将数据集划分为训练集和测试集。接着，使用K-Means算法对训练集进行聚类，并为每个簇分配一个类别标签。随后，计算分类精确度和误差率，并生成CSV文件记录每个簇的标签分布情况。同时，生成分类报告，包含精确度、误差率、数据集大小等信息。最终，通过去除混淆矩阵可视化部分，重点输出分类结果和性能指标。

**主要超参数解释**

以下是代码中的几个关键超参数，并详细说明它们的作用和影响：

1. k = 60

作用：该参数表示K-Means聚类算法中簇的数量，即将所有样本分成60个簇。

影响：选择合适的簇数对聚类结果有重要影响。如果簇数过多，可能导致每个簇的样本过少，模型可能无法有效区分类别。如果簇数过少，则可能将多个不同类别的样本聚集在一起，导致分类效果不佳。

调整建议：通常使用肘部法（Elbow Method）或轮廓系数（Silhouette Score）来选择最优的簇数。

2. n\_init = 10

作用：K-Means算法中的初始化次数，表示K-Means算法在执行时会进行10次不同的初始化，并返回最好的结果。

影响：K-Means对初始簇中心的选择较为敏感，通过增加初始化次数可以避免局部最优解，从而提高聚类效果。

调整建议：一般选择n\_init=10，如果运行时间较长且数据量较小，可以适当减少。

# DBSCAN算法

## 算法发展与思路

### 1.1 算法介绍

\*\*DBSCAN（Density-Based Spatial Clustering of Applications with Noise）\*\*是一种基于密度的聚类算法，广泛应用于数据挖掘和机器学习领域。与传统的聚类方法（如K-means和层次聚类）不同，DBSCAN不需要预先指定簇的数量，而是通过数据点的密度来自动发现聚类结构。其优势在于能够识别形状不规则、密度不同的聚类，并且能够识别噪声点，避免了将噪声错误地分配到簇中。DBSCAN的提出解决了K-means算法要求事先指定簇数量、假设簇形状为球形且簇密度均匀等局限性，特别适合于处理复杂和非均匀密度的数据集。

DBSCAN由Martin Ester、Hans-Peter Kriegel、Jörg Sander和Xiaowei Xu于1996年提出[[7]](#footnote-6)。这篇论文首次提出了DBSCAN的核心思想，并发表在国际数据库学会（ACM SIGMOD）会议上。提出DBSCAN的背景是当时空间数据（如地理位置数据和图像数据）聚类的需求越来越迫切，而传统的聚类方法无法有效处理这些数据的高维性、稀疏性和形状不规则性。

DBSCAN的核心思想是通过点的密度来定义聚类，而非依赖于事先设定簇的数量。算法通过两个重要参数：ε（邻域半径）和MinPts（邻域内最少点数）来判断点是否属于核心点、边界点或噪声点。若一个点的ε邻域内包含至少MinPts个点，则该点为核心点，密度高的区域即为聚类。边界点是邻域内点数少于MinPts但与核心点相连的点，噪声点则是既不是核心点也不在任何核心点邻域中的点。

DBSCAN的优势在于其自动发现聚类数量、处理复杂簇形状的能力，以及识别噪声的能力。它能够发现形状不规则的簇，不像K-means只能识别球形簇。此外，DBSCAN还能有效处理含噪声数据，避免噪声点被错误地分配到簇中，因此广泛应用于图像处理、空间数据分析和异常检测等领域。

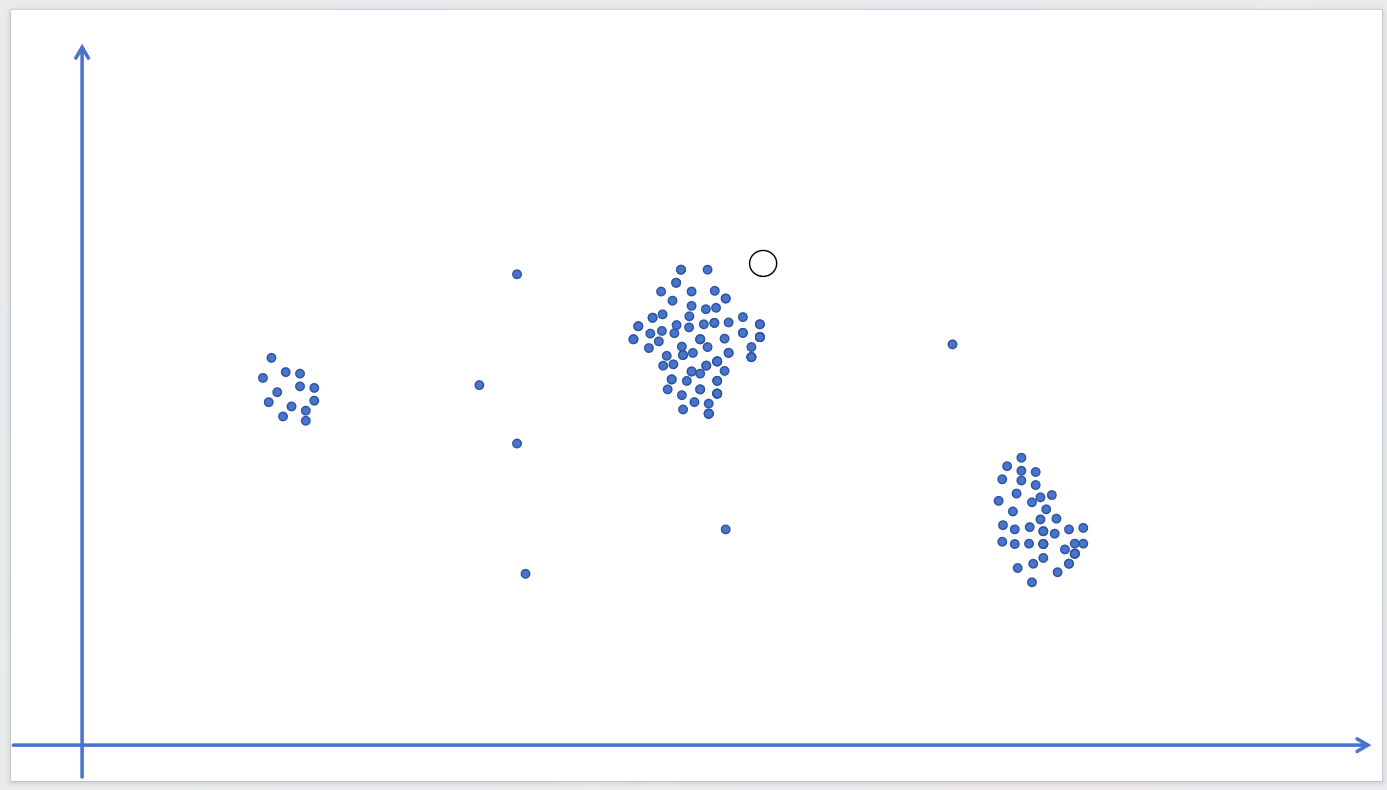
自提出以来，DBSCAN在多个领域取得了显著的应用成果。例如，在地理信息系统（GIS）中，DBSCAN被用于地理位置数据的聚类，识别热点区域；在图像处理中，尤其是在医学图像分析中，DBSCAN被用于分割不规则形状的区域；在异常检测中，由于DBSCAN能识别噪声点，它被广泛应用于金融欺诈检测和设备故障诊断等场景。此外，DBSCAN在大数据和高维数据的聚类任务中提供了比传统方法更灵活、鲁棒的解决方案，尤其适用于稀疏数据和异质数据的聚类。

总体来说，DBSCAN算法为处理复杂数据集提供了强大的工具，尤其适用于含噪声、形状不规则且密度不同的聚类任务，推动了密度聚类方法的研究并影响了后续相关技术的发展。

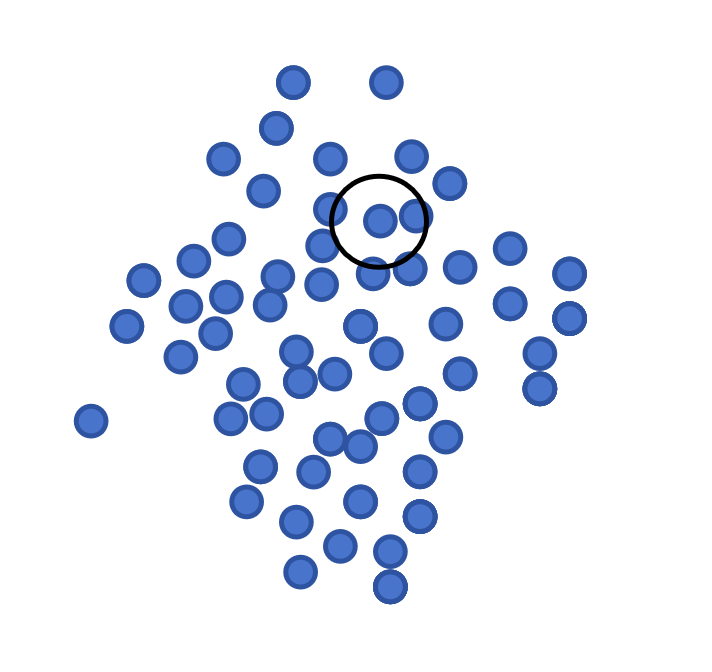
### 1.2 算法思想

在上文中我们已经简略提到了DBSCAN算法的思想以及俩个重要参数：领域以及领域内最小点数。那么DBSCAN如何使用这两个参数来工作呢？

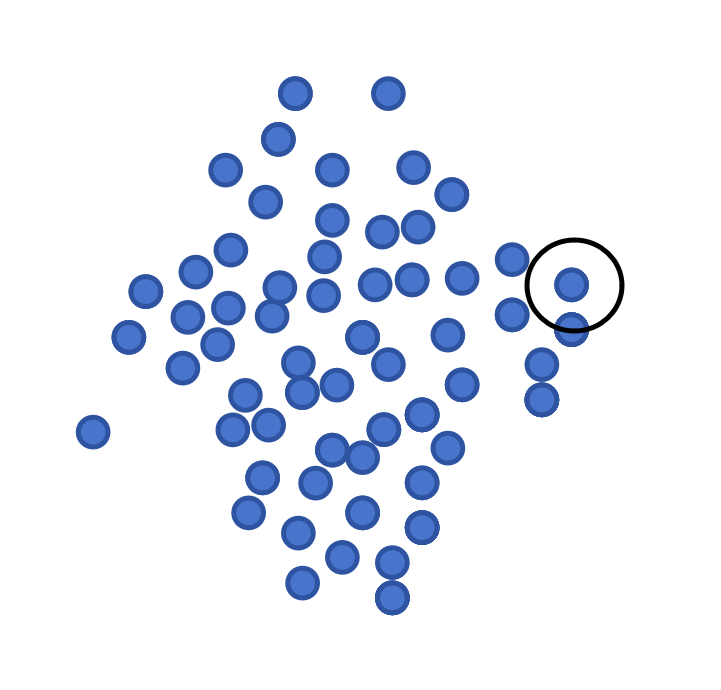
首先假设我们有数个点数据点如下：



图中蓝色的点即为现在的未分类数据，黑色的圈即为我们所定义的用来寻找核心点和边界点[[8]](#footnote-7)的，其半径即为。在进行选择的时候，我们每一次以任意的数据点作为圆心，来检查是否属于核心点。我们规定，那么在范围内包含三个点的才可以认为是核心点：

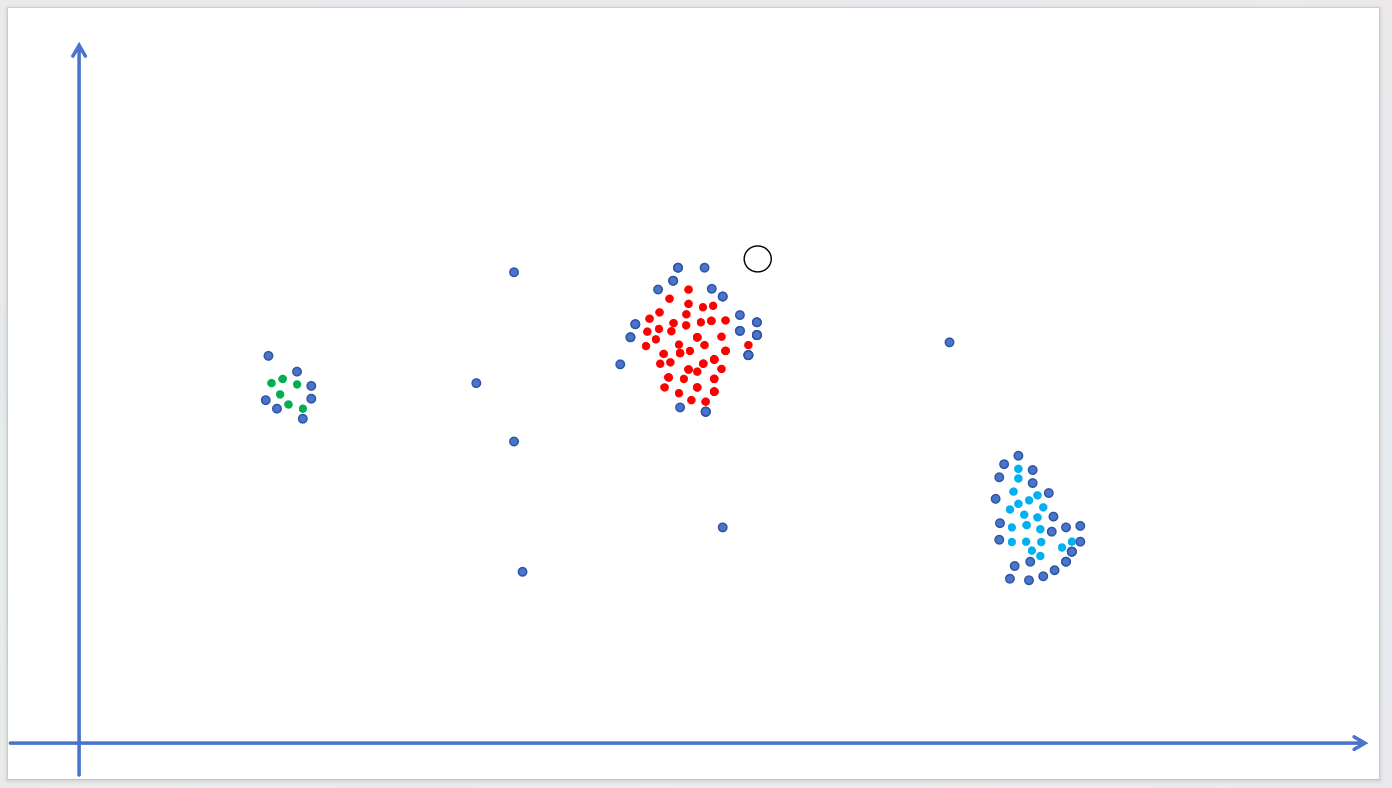


如图所示，这一点的范围内包含5个数据的，那么我们可以认为这个点为核心点。



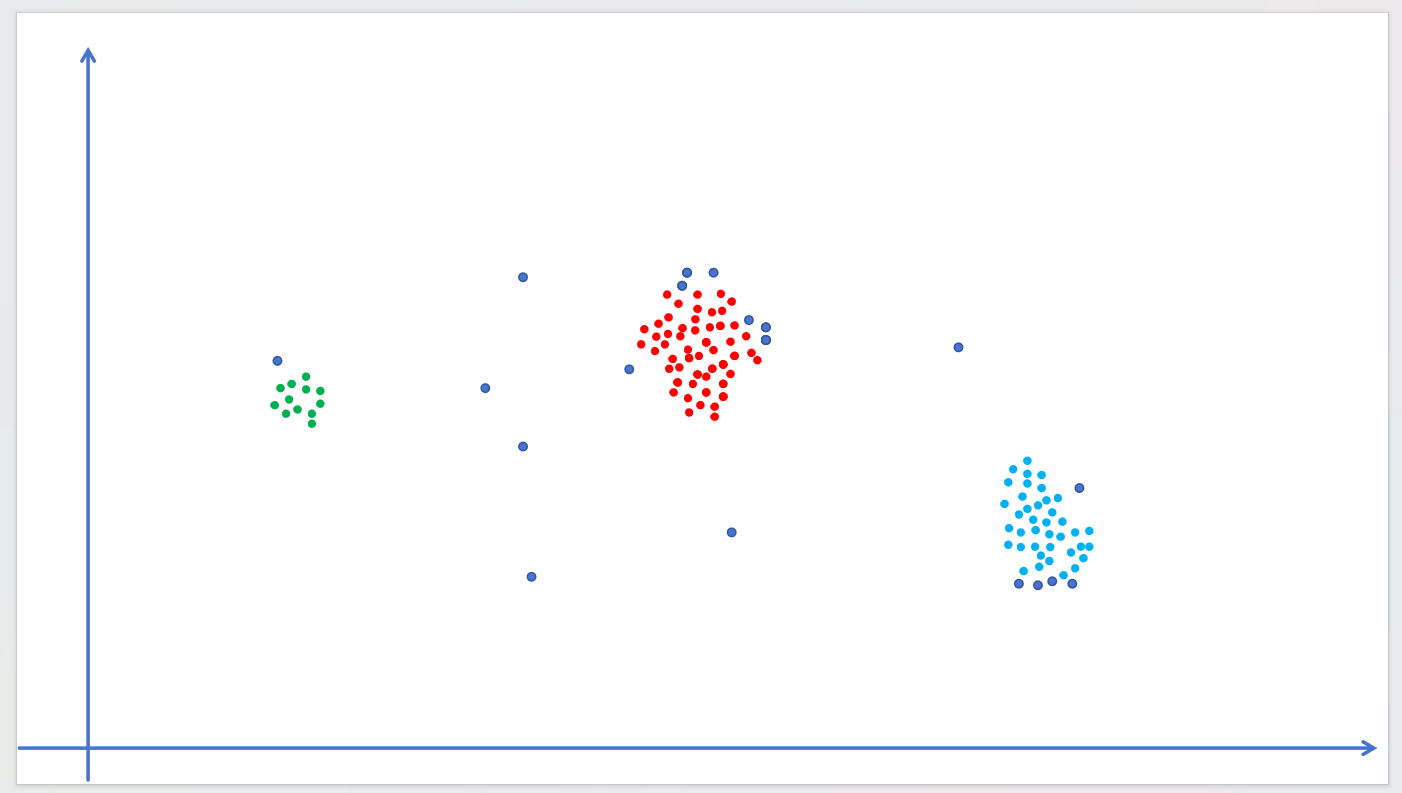
这一点的范围内仅包含一个点，那么这个点就是边界点。

经过检查以后我们可以得到如下图所示：



可以看到，在第一轮筛选过后，我们找到了所有的核心点并分类了簇，根据这些核心点，我们开始进一步扩展边界点

最终经过扩展，我们可以得到核心点与边界点如图所示：



上图中，深蓝色的即为噪点，红、绿浅蓝代表不同的种类。

## 2.DBSCAN算法数学原理

### 2.1 核心概念与数学定义

#### 2.1.1 距离的度量

在DBSCAN算法中，我们度量距离使用欧式距离来进行计算，对于任意的两个点、，有：

#### 2.1.2 **-领域**

对于数据集D中的任意一点p，其-邻域定义为在距离度量d下，距离p不超过的所有点的集合：

#### 2.1.3 密度可达

对于两个点和，如果存在一个点序列使得：

且对于所有，都有：

则称q从p密度可达，记作。

#### 2.1.4 密度连接

对于任意两个点p和q，如果存在一个点o使得：

则称p和q密度连接，记作。

密度可达与密度连接是DBSCAN算法中确保聚类准确性和连贯性的核心概念。**密度可达**定义了点与点之间通过一系列密集核心点相互连接的路径，确保簇的扩展基于数据点的局部密度连续性。而**密度连接**则确保任意两个点如果能够通过至少一个共同的核心点相互密度可达，则它们属于同一个簇。这两个概念共同作用，使得DBSCAN能够识别任意形状的簇、有效处理噪声数据，并自动确定簇的数量，从而实现基于密度的连通性聚类。

### 2.2 密度连接图的构建与证明

#### 2.2.1 密度图的构建

考虑数据集，我们可以基于一个密度连通图，其中：

**顶点集：**

边集：

即，如果p和q密度连接，则在G中存在一条边连接它们。

#### 2.2.2 密度连接形成连通分量证明

**目标：**证明密度连接关系是一个等价关系，从而将G分解为若干个连通分量，每个连通分量对应一个簇。

**等价关系的三大性质**：

要证明是等价关系，需要验证以下三点：

1.自反性：对于任意点，需要。

证明：选择。由于（可以选择序列长度为1），则：

因此，自反性成立。

2.对称性：如果，则.

证明：由定义，存在o使得：

同样存在使得：

因此：

对称性成立。

3.传递性：如果且，则

证明：由，存在使得：

由，存在使得：

由于q是从p和r密度可达的桥梁，可以构建从p到r的密度可达路径。例如，通过或，满足：

从而：

传递性成立。

由于满足自反性、对称性和传递性，它是一个等价关系。因此，集合D可以被划分成若干个等价类，每个等价类对应一个簇。

### 2.3 簇的数学表达及连通性证明

#### 2.3.1 簇的数学定义与算法实现

簇的定义：根据等价关系划分，簇定义为：

每个是的一个连通分量。

簇的算法实现伪代码：

DBSCAN(data, epsilon, MinPts):

# 初始化所有点的状态

for each point p in data:

p.status = 'unvisited' # 标记为未访问

p.cluster = -1 # 表示尚未分配到任何簇

cluster\_id = 0 # 初始化簇编号

# 遍历数据点

for each point p in data:

if p.status == 'unvisited': # 如果 p 还未被访问过

p.status = 'visited' # 标记 p 为已访问

N\_eps = regionQuery(p, epsilon) # 计算 p 的 ε-邻域

if size(N\_eps) >= MinPts: # 如果 p 是核心点

cluster\_id += 1 # 新建一个簇

expandCluster(p, N\_eps, cluster\_id, epsilon, MinPts) # 扩展簇

else:

p.status = 'noise' # 如果不是核心点，标记为噪声点

# 扩展簇的方法

expandCluster(p, N\_eps, cluster\_id, epsilon, MinPts):

p.cluster = cluster\_id # 将 p 分配到当前簇

# 创建一个队列或列表，用于处理簇内的点

seeds = N\_eps # 将 p 的邻域内的点作为种子

for each point q in seeds:

if q.status == 'unvisited': # 如果 q 未访问

q.status = 'visited' # 标记 q 为已访问

N\_q = regionQuery(q, epsilon) # 计算 q 的 ε-邻域

if size(N\_q) >= MinPts: # 如果 q 是核心点

seeds = seeds ∪ N\_q # 将 q 的邻域内的点添加到种子集

if q.cluster == -1: # 如果 q 没有被分配到任何簇

q.cluster = cluster\_id # 将 q 分配到当前簇

# 计算点 p 的 ε-邻域

regionQuery(p, epsilon):

N\_eps = [] # 存储 p 的 ε-邻域点

for each point q in data:

if distance(p, q) <= epsilon: # 如果 q 在 p 的 ε-邻域内

N\_eps.append(q)

return N\_eps

#### 2.3.2 簇的连通性证明

为了证明通过密度连接形成的簇是一个连通分量，考虑以下几点：

1.**任意两个点在同一簇内是密度连接的**：对于任意的，由密度连接的定义，存在一个共同的点o使得：

这意味着存在从p到o和从q到o的密度可达路径，因此存在从p到q的路径，保证了连通性。

1. **连通分量的最大性：**如果存在一个，且与中的某个点p密度连接，则r应该属于，则与矛盾。因此，每个连通分量是最大化的，无法在扩展。

## DBSCAN算法代码实现

使用sklearn库实现DBSCAN算法如下：

import os

import warnings

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from PIL import Image

from sklearn.model\_selection import train\_test\_split

from sklearn.cluster import DBSCAN

from sklearn.metrics import adjusted\_rand\_score, silhouette\_score

from sklearn.neighbors import NearestNeighbors

from scipy.spatial.distance import cdist

from scipy.optimize import linear\_sum\_assignment

from sklearn.preprocessing import LabelEncoder

import torch

from torch.utils.data import Dataset, DataLoader

from torchvision import models, transforms

from tqdm import tqdm # For progress bars

# Suppress unnecessary warnings

warnings.filterwarnings("ignore")

# ================================

# Section 1: GPU Setup

# ================================

def setup\_gpu(device\_id=0):

"""

Configure the GPU device for PyTorch. If a GPU is available, it sets the specified GPU as the device.

Otherwise, it defaults to CPU.

Args:

device\_id (int): The ID of the GPU to use.

Returns:

torch.device: The configured device.

"""

if torch.cuda.is\_available():

device = torch.device(f'cuda:{device\_id}')

print(f"GPU {device\_id} is available and set as the device.")

else:

device = torch.device('cpu')

print("No GPU detected. Using CPU.")

return device

# ================================

# Section 2: Data Loading and Preprocessing

# ================================

def load\_data(data\_dir):

"""

Load image file paths and their corresponding labels from the specified directory.

The directory should contain subdirectories, each representing a class label.

Args:

data\_dir (str): Path to the data directory.

Returns:

pd.DataFrame: DataFrame containing file paths and labels.

"""

data = []

labels = []

for folder\_name in os.listdir(data\_dir):

folder\_path = os.path.join(data\_dir, folder\_name)

if os.path.isdir(folder\_path):

# Extract label: first character after the first underscore

parts = folder\_name.split('\_')

if len(parts) > 1 and len(parts[1]) > 0:

label = parts[1][0]

else:

label = 'unknown'

for img\_name in os.listdir(folder\_path):

if img\_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):

img\_path = os.path.join(folder\_path, img\_name)

data.append(img\_path)

labels.append(label)

df = pd.DataFrame({

'filepath': data,

'label': labels

})

return df

# ================================

# Section 3: Feature Extraction using VGG16

# ================================

class ImageDataset(Dataset):

"""

Custom Dataset for loading images from file paths.

"""

def \_\_init\_\_(self, filepaths, transform=None):

"""

Initialize the dataset with file paths and optional transformations.

Args:

filepaths (list): List of image file paths.

transform (callable, optional): Optional transform to be applied on a sample.

"""

self.filepaths = filepaths

self.transform = transform

def \_\_len\_\_(self):

"""

Return the total number of samples.

"""

return len(self.filepaths)

def \_\_getitem\_\_(self, idx):

"""

Retrieve an image and apply transformations.

Args:

idx (int): Index of the sample to retrieve.

Returns:

torch.Tensor: Transformed image tensor.

"""

img\_path = self.filepaths[idx]

try:

image = Image.open(img\_path).convert('RGB')

except Exception as e:

print(f"Failed to load image {img\_path}: {e}")

# Return a black image to maintain batch size consistency

image = Image.new('RGB', (224, 224))

if self.transform:

image = self.transform(image)

return image

def extract\_features\_pytorch(filepaths, model, transform, device, batch\_size=32):

"""

Extract features from images using a pre-trained PyTorch model.

Args:

filepaths (list): List of image file paths.

model (torch.nn.Module): Pre-trained PyTorch model for feature extraction.

transform (torchvision.transforms.Compose): Transformations to apply to the images.

device (torch.device): Device to perform computations on.

batch\_size (int): Number of samples per batch.

Returns:

np.ndarray: Extracted feature vectors.

"""

dataset = ImageDataset(filepaths, transform=transform)

dataloader = DataLoader(dataset, batch\_size=batch\_size, shuffle=False,

num\_workers=os.cpu\_count() if os.cpu\_count() > 0 else 0)

features = []

with torch.no\_grad():

for batch in tqdm(dataloader, desc="Extracting Features"):

batch = batch.to(device)

output = model(batch)

output = output.cpu().numpy()

features.append(output)

features = np.vstack(features)

return features

# ================================

# Section 4: Clustering Evaluation

# ================================

def cluster\_accuracy\_scipy(y\_true, y\_pred):

"""

Calculate clustering accuracy using the Hungarian algorithm to map predicted clusters to true labels.

Args:

y\_true (array-like): True labels.

y\_pred (array-like): Predicted cluster labels.

Returns:

float: Clustering accuracy.

"""

# Exclude noise points

mask = y\_pred != -1

y\_true\_filtered = y\_true[mask]

y\_pred\_filtered = y\_pred[mask]

if len(y\_pred\_filtered) == 0:

return 0.0

# Encode labels as integers

le\_true = LabelEncoder()

le\_pred = LabelEncoder()

y\_true\_enc = le\_true.fit\_transform(y\_true\_filtered)

y\_pred\_enc = le\_pred.fit\_transform(y\_pred\_filtered)

# Create confusion matrix

matrix = np.zeros((y\_pred\_enc.max() + 1, y\_true\_enc.max() + 1), dtype=int)

for i in range(y\_pred\_enc.size):

matrix[y\_pred\_enc[i], y\_true\_enc[i]] += 1

# Apply Hungarian algorithm for optimal matching

row\_ind, col\_ind = linear\_sum\_assignment(-matrix) # Maximizing the total matches

# Calculate accuracy

accuracy = matrix[row\_ind, col\_ind].sum() / y\_pred\_enc.size

return accuracy

def assign\_cluster(test\_sample, core\_features, core\_labels, epsilon):

"""

Assign a test sample to the nearest cluster based on epsilon distance.

Args:

test\_sample (np.ndarray): Feature vector of the test sample.

core\_features (np.ndarray): Feature vectors of core samples from training data.

core\_labels (np.ndarray): Cluster labels of core samples.

epsilon (float): Maximum distance to consider for cluster assignment.

Returns:

int: Assigned cluster label or -1 if considered noise.

"""

distances = cdist([test\_sample], core\_features, metric='euclidean')[0]

min\_dist = np.min(distances)

if min\_dist <= epsilon:

cluster\_id = core\_labels[np.argmin(distances)]

return cluster\_id

else:

return -1

# ================================

# Section 5: Main Execution Flow

# ================================

def main():

"""

Main function to execute the clustering pipeline:

1. Setup GPU.

2. Load and preprocess data.

3. Extract features using VGG16.

4. Split data into training and testing sets.

5. Perform DBSCAN clustering on training data.

6. Evaluate clustering performance.

7. Assign clusters to test data and evaluate.

"""

# Step 1: Setup GPU

device = setup\_gpu(device\_id=0) # Change device\_id if multiple GPUs are available

# Step 2: Load Data

data\_dir = 'need' # Replace with your actual data directory

df = load\_data(data\_dir)

print(f"Total samples: {len(df)}")

print(df.head())

# Step 3: Define Image Transformations

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], # VGG16 pre-trained mean

std=[0.229, 0.224, 0.225]) # VGG16 pre-trained std

])

# Step 4: Load Pre-trained VGG16 Model for Feature Extraction

vgg16 = models.vgg16(weights=models.VGG16\_Weights.IMAGENET1K\_V1)

vgg16.classifier = torch.nn.Identity() # Remove the classification layers

vgg16.to(device)

vgg16.eval()

# Step 5: Extract Features

print("Extracting features from images...")

X = extract\_features\_pytorch(df['filepath'].tolist(), vgg16, transform, device)

y = df['label'].values

print(f"Feature dimensions: {X.shape}")

# Step 6: Split Data into Training and Testing Sets

X\_train, X\_test, y\_train, y\_test, paths\_train, paths\_test = train\_test\_split(

X, y, df['filepath'], test\_size=0.2, random\_state=42, stratify=y

)

print(f"Training samples: {len(X\_train)}")

print(f"Testing samples: {len(X\_test)}")

# Step 7: Plot K-Distance Graph to Determine Epsilon

def plot\_k\_distance(X, k):

"""

Plot the K-distance graph to help determine the epsilon parameter for DBSCAN.

Args:

X (np.ndarray): Feature vectors.

k (int): Number of nearest neighbors.

"""

nbrs = NearestNeighbors(n\_neighbors=k).fit(X)

distances, \_ = nbrs.kneighbors(X)

k\_distances = np.sort(distances[:, k - 1])

plt.figure(figsize=(10, 6))

plt.plot(k\_distances)

plt.xlabel('Sample Index')

plt.ylabel(f'{k}th Nearest Neighbor Distance')

plt.title(f'K-Distance Graph (k={k})')

plt.grid(True)

plt.show()

# Plot K-distance graph with MinPts=4

plot\_k\_distance(X\_train, k=4)

# Step 8: Set DBSCAN Parameters

# NOTE: The epsilon value should be chosen based on the K-distance graph.

# Here, we assume the elbow is around 0.3. Adjust as necessary.

epsilon = 0.3

min\_samples = 4

# Step 9: Train DBSCAN Model

print("Training DBSCAN model...")

dbscan = DBSCAN(eps=epsilon, min\_samples=min\_samples, metric='euclidean', n\_jobs=-1)

dbscan.fit(X\_train)

labels\_train = dbscan.labels\_

num\_clusters = len(set(labels\_train)) - (1 if -1 in labels\_train else 0)

num\_noise = list(labels\_train).count(-1)

print(f"Number of clusters (excluding noise): {num\_clusters}")

print(f"Number of noise points: {num\_noise}")

# Step 10: Evaluate Clustering on Training Data

# Adjusted Rand Index (ARI)

ari = adjusted\_rand\_score(y\_train, labels\_train)

print(f"Adjusted Rand Index (ARI) on Training Data: {ari:.4f}")

# Silhouette Score (only if number of clusters > 1)

if num\_clusters > 1:

silhouette = silhouette\_score(X\_train, labels\_train)

print(f"Silhouette Score on Training Data: {silhouette:.4f}")

else:

print("Silhouette Score cannot be computed with less than 2 clusters.")

# Clustering Accuracy

train\_accuracy = cluster\_accuracy\_scipy(y\_train, labels\_train)

print(f"Clustering Accuracy on Training Data: {train\_accuracy:.4f}")

# Step 11: Assign Clusters to Test Data

print("Assigning clusters to test data...")

# Identify core samples from training data

core\_samples\_mask = np.zeros\_like(dbscan.labels\_, dtype=bool)

core\_samples\_mask[dbscan.core\_sample\_indices\_] = True

core\_features = X\_train[core\_samples\_mask]

core\_labels = labels\_train[core\_samples\_mask]

labels\_test = []

for idx, sample in tqdm(enumerate(X\_test), total=len(X\_test), desc="Assigning Clusters"):

cluster\_id = assign\_cluster(sample, core\_features, core\_labels, epsilon)

labels\_test.append(cluster\_id)

labels\_test = np.array(labels\_test)

# Step 12: Evaluate Clustering on Test Data

# Adjusted Rand Index (ARI)

ari\_test = adjusted\_rand\_score(y\_test, labels\_test)

print(f"Adjusted Rand Index (ARI) on Test Data: {ari\_test:.4f}")

# Silhouette Score (only if number of clusters > 1 and less than number of samples)

unique\_labels\_test = set(labels\_test)

if len(unique\_labels\_test) > 1 and len(unique\_labels\_test) < len(X\_test):

silhouette\_test = silhouette\_score(X\_test, labels\_test)

print(f"Silhouette Score on Test Data: {silhouette\_test:.4f}")

else:

print("Silhouette Score cannot be computed due to insufficient number of clusters.")

# Clustering Accuracy

test\_accuracy = cluster\_accuracy\_scipy(y\_test, labels\_test)

print(f"Clustering Accuracy on Test Data: {test\_accuracy:.4f}")

# Optionally, save the results to a CSV file

results\_df = pd.DataFrame({

'filepath': paths\_test,

'true\_label': y\_test,

'predicted\_cluster': labels\_test

})

results\_df.to\_csv('clustering\_results.csv', index=False)

print("Clustering results saved to 'clustering\_results.csv'.")

# ================================

# Entry Point

# ================================

if \_\_name\_\_ == '\_\_main\_\_':

main()

**超参数讲解：**

epsilon (DBSCAN的邻域半径)

选择标准：根据K-距离图的“肘部”位置来选择epsilon值。在代码中，epsilon默认值为0.3。

作用：定义样本之间的最大距离，超过这个距离的样本将被认为属于不同的聚类。

min\_samples (DBSCAN的最小样本数)

默认值：4

作用：定义形成一个聚类所需的最小样本数。较小的min\_samples值可能导致更多的噪声点。

k (K-距离图的K值)

默认值：4

作用：用于K-距离图中寻找第K近邻的距离，这个值帮助确定合适的epsilon值。

1. J. MacQueen, "Some methods for classification and analysis of multivariate observations," Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281-297, 1967. [↑](#footnote-ref-0)
2. 2.2节将详细介绍。 [↑](#footnote-ref-1)
3. 簇点随机生成，后续需要不断更新，详细会在3.3节展示。 [↑](#footnote-ref-2)
4. 在聚类算法中，要求簇点为质心，即簇内数据中心点。 [↑](#footnote-ref-3)
5. 式子中a.s.代表几乎必然。 [↑](#footnote-ref-4)
6. 在这里一般会随机选择任意数据作为簇的质心（即簇点），得到的簇的质心就是数据的参数，会在后续进行更新。 [↑](#footnote-ref-5)
7. M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996, pp. 226-231. [↑](#footnote-ref-6)
8. 需要注意的是，核心点是可以扩散的，但是边界点不可以，在会在后文中详细表示。 [↑](#footnote-ref-7)